Heterogeneous Service Selection based on Formal Concept Analysis

Stéphanie Chollet, Vincent Lestideau, Philippe Lalanda, Diana Moreno-Garcia
Laboratoire Informatique de Grenoble
F-38041, Grenoble cedex 9, France
(Stephanie.Chollet, Vincent.Lestideau, Philippe.Lalanda, Diana-Guadalupe.Moreno-Garcia)@imag.fr

Pierre Colomb
Laboratoire d'Informatique, de Modélisation et d'Optimisation des Systèmes
F-63173 AUBIERE cedex, France
pierre.colomb@univ-bpclermont.fr

Abstract

In this paper, we propose an approach based on Formal Concept Analysis in order to organize the services registry at runtime and to allow the "best" service selection among heterogeneous and secured services according to a set of specifications. This solution has been validated in the European SODA project on pervasive applications.

1 Introduction

Pervasive computing is today a research field of major importance. This computing domain is actually changing the way we envisage our environment: it relies on smart, communication-enabled devices transparently interacting with us in our living spaces. These devices tend to disappear in the environment: they are numerous but not always perceivable. At least, their computing capabilities are not always apparent. In smart buildings, for instance, many devices are integrated in the physical infrastructure in order to offer us a number of advanced services related to comfort, caretaking, etc.

Devices thus communicate more and more with each other, configure or repair themselves, and perform context-aware cognitive and physical actions. The vision of coordinated or cooperating devices teaming up transparently to provide advanced services of all sorts is actually getting closer and closer. However, lots of research programs in pervasive computing have focused on hardware and communication protocols, wireless or not. Consequently, plenty of devices are today available and commercialized but they often stay isolated in "computing islands" and very few integrated services are actually proposed. We believe that a solid software infrastructure is needed to accomplish the pervasive service vision. Building such an infrastructure is a complex, often underestimated, task. Indeed, several software engineering challenges remain to be tackled before fulfilling the vision of a true pervasive world. Notably the high degree of dynamism, distribution, heterogeneity and autonomy of the devices involved raises important problems. Once again, the building environment perfectly illustrates the targeted environment and the implied software complexity. It is open to dynamic connections: devices may enter and leave the network spontaneously, providing context-dependent features (e.g. depending on user's activity). It is also open to heterogeneous devices: protocols and device types differ according to application domains and providers.

Service-Oriented Computing (SOC) [18] is today a solution of choice to implement dynamic devices and applications in pervasive environments. This new paradigm appeared a few years ago. The very purpose of this reuse-based approach is to build applications through the late composition of independent software elements, called services. Services are described and published by service providers; they are chosen and invoked by service consumers. SOC thus support dynamic service discovery and lazy inter-service binding. Such characteristics are essential when building pervasive applications with strong adaptability requirements. A key point is the ability to select at anytime the "best" service available to realize an application. Many mechanisms, often formal, have been defined to do so [13]. An important aspect which has not been studied that much is the organization of the services registry, that is the place where services specifications are dynamically stored and updated. We believe that, to tackle the building requirements, specific mechanisms are needed to ensure an efficient access to available services and to support decision making in evolving environment.

In this paper, we investigate a solution based on Formal Concept Analysis (FCA) in order to organize the services registry at runtime and to allow the ”best” service selection among heterogeneous and secured services according to a set of specifications. This solution, that has been tested with the industrial partners of the European SODA[1] project, brings significant results in terms of efficiency and adaptability. The paper is organized as follows. First, some background about heterogeneous service composition and existing tools adaptated to this composition are provided. Section 3 presents our approach, based on the theoretical foundation of Formal Concept Analysis, and its application to the service domain. Before the related work and the conclusion, Section 4 presents the experiments and results obtained in the case of an alarm system.

2 Background

2.1 Service-Oriented Architecture

Service-oriented computing brings software qualities of major importance. As with any planned reuse approach, it supports rapid, high quality development of software applications. Using existing, already tested, software elements is likely to reduce the time needed to build up an application and improve its overall quality. Weak coupling between consumers and providers reduces dependencies among composition units, letting each element to evolve separately. Late binding and substitutability improve adaptability: a service chosen or replaced at runtime, based on its current availability and properties, is likely to better fulfill the consumer expectations.

	Web Service	**DPWS**	**UPnP**
Publish Discovery	UDDI Registry	Multicast	Multicast
Service Description	WSDL	Extended WSDL	UPnP Device Description
Composition	WS-BPEL	-	-

Table 1. Comparison of service technologies.

Not surprisingly, a number of implementations have been proposed, sometimes for different purposes. Web Services (www.w3c.org), for instance, represent a solution of choice for software integration. UPnP (www.upnp.org) and DPWS [25, 11] (Devices Profile for Web Services) are heavily used in pervasive applications in order to implement volatile devices. OSGi [17] (www.osgi.org) and iPOJO [5] (www.ipojo.org) provide advanced dynamic features advantageously used to build pervasive gateways. As illustrated by Table 1, different technologies use different description languages, different notification styles, and different invocation styles.

The integration of non-functional features is another serious challenge when building service-oriented applications. In particular, security acts today as a brake to the massive adoption of services. In some distributed environments, software services can be alarmingly vulnerable and organizations are exposed to a considerable amount of security risk and dependability degradation. This is why, in the last few years, many security protocols for Web Services, such as XML Digital Signature [23], XML Encryption [22], SAML [15], XACML [16] and the WS-* ([14], [9],...) have been proposed to enable secure Web Services calls.

Integrating heterogeneous, dynamic and secured services is thus admittedly complex. As illustrated by Figure 1, this is exactly what is required in pervasive domains. In this field, applications frequently need to integrate UPnP-based and DPWS-based field devices and Web Services for remote applications. These services are secured using different techniques and technologies. In addition, most services are dynamic: smart devices join and leave the network at unpredictable times; back office applications are regularly updated.

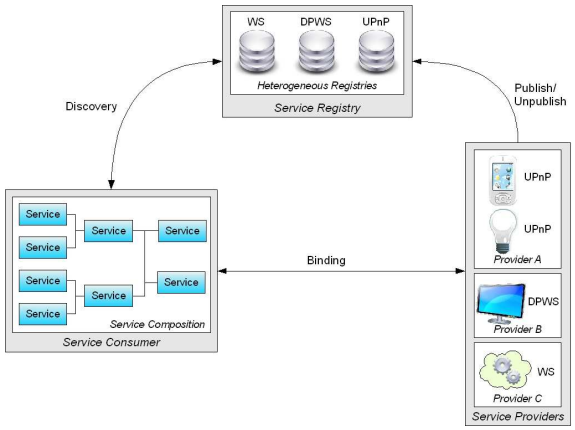

Figure 1. Heterogeneous services.

In order to select a service at runtime, a number of characteristics have to be considered. Of course, functional compatibility is essential. But, a composition has also to meet a set of transverse non-functional qualities like availability, security or cost effectiveness. It may also consider the available implementation technologies. In domains like pervasive, the number of available services at a given moment can be high and it becomes important to find mechanisms allowing efficient multi-criteria selection.

[1] SODA is a European project partly funded by the French Ministry of industry and brings together, among others, Schneider Electric, Thales and Grenoble University.

2.2 Existing tools

Many tools have been developed to facilitate the design and execution of applications made of dynamic, heterogeneous, secured services [4], [12], [24]. One of the solutions to hide the complexity of technology heterogeneity is to adopt a model-driven approach. The environment provides users with a workplace hiding technical details and letting them define service composition at a level of abstraction in line with their concerns. This abstract composition is then incrementally transformed into an executable one.

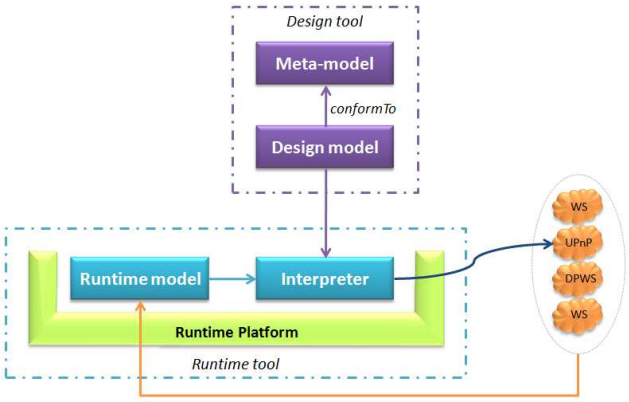

Figure 2. Existing approach.

This model-driven approach, illustrated on Figure 2, is based on the notion of abstract services and concrete services. An abstract service is defined in the following terms:

- A signature defining the *id* of the service and its inputs and outputs in terms of products. This part corresponds to WSDL or SCPD extracts for instance, if dealing with Web services, respectively UPnP services. Extracts only contain implementation-independent information.
- Target-technologies implementing the abstract service. The expert designer knows, at design time, which technologies can implement the required functionalities.
- A set of properties, mandatory or desirable. Among them, security properties play a major role.

A concrete service is an implementation of an abstract service in a given service technology. For instance, a concrete service can be a Web Service or a UPnP service. Several concrete services can be made available for a single abstract service.

The runtime model maintains a view of the environment to select appropriate and available concrete services. It is an advanced registry containing the services technologies, the services functional characteristics, the security characteristics provided/required according to the server point of view and other properties like cost, reputation, response time... The concrete services are stored in a services registry. One of the challenges for the runtime model is its evolution during execution because services arrive and depart at anytime. The runtime model must maintain a coherent view of the execution environment.

The concrete services of the registry can be selected with brute-force algorithms when the number of available services and constraints are weak. But, this type of algorithms is inefficient if many services are stored in the registry, such as in the smart building domain. The purpose of this paper is to provide a new approach to better structure the runtime model and to improve decision making.

3 Approach

In order to better structure the services registry, we studied and adapted the Formal Concept Analysis (FCA) approach to service-oriented computing. First, we provide an introduction to FCA.

3.1 Theoretical foundations

FCA [8] is a well-known classification tool used in many practical cases [20], [2], [21]. The purpose of this approach is to build a partially ordered structure, called concept lattice, from a formal context.

A **formal context** $\mathbb{K}$ is a set of relations between objects and attributes. It is denoted by $\mathbb{K} = (O, A, R)$ where O and A are respectively sets of Objects and Attributes, and R is a relation between O and A. As an example, Table 2 illustrates a formal context with $O = \{1, 2, 3, 4\}$ and $A = \{a, b, c, d\}$.

	a	b	c	d
1	x	x	x	
2		x	x	x
3	x	x		
4			x	x

Table 2. Context example.

A **formal concept** C is a pair (E, I) where E is a set of objects called Extent, I is a set of attributes called Intent, and all the objects in E are in relation R with all the attributes in I. Thus, the Extent of a concept is the set of **all** objects sharing a set of common attributes, and the Intent is the set of **all** attributes shared by the objects of the Extent. Formally:

- $E = \{o \in O \mid \forall i \in I, (o, i) \in R\}$,

- $I = \{a \in A \mid \forall e \in E, (e, a) \in R\}$.

Consequently, a concept $C = (E, I)$ is made of the objects in E which are exactly the set of objects sharing the attributes in I. In the previous example, $(2, 4; c, d)$ is a concept. Indeed, objects 2 and 4 share the attributes c and d, and no other object share c, d as attributes. Contrarily, we can verify on Table 2 that $(1, 2, 4; b, c)$ is not a concept because object 4 does not own attribute b. Similarly, the attributes a and b are shared by the objects 1 and 3 therefore $(1; a, b)$ is not a concept.

The set $\mathcal{C}(\mathbb{K})$ of all concepts induced by a context can be ordered using the following partial order relation: $(E_1, I_1) \leq (E_2, I_2) \Leftrightarrow E_2 \subseteq E_1 \; and \; I_1 \subseteq I_2$. (E_2, I_2) is defined as a successor of (E_1, I_1).

A **concept lattice** is defined as the ordering $\mathcal{C}(\mathbb{K})$ with $\leq$. It can be represented by a particular graph called Hass Diagram as illustrated in Figure 3.

A concept (E_2, I_2) is an immediate successor of a concept (E_1, I_1) if the following conditions hold :

1. $(E_1, I_1) \leq (E_2, I_2)$,
2. There is no concept (E_3, I_3) such that $(E_1, I_1) \leq (E_3, I_3) \leq (E_2, I_2)$.

Intuitively, immediate successors of a concept (E, I) are concepts located just above (E, I) in the Hass diagram of the concept lattice. Given a concept (E, I) we denoted by $Succ((E, I))$ the set of all its immediate successors. As an example $Succ((1, 2, 3; b)) = \{(1, 3; a, b); (1, 2; b, c)\}$

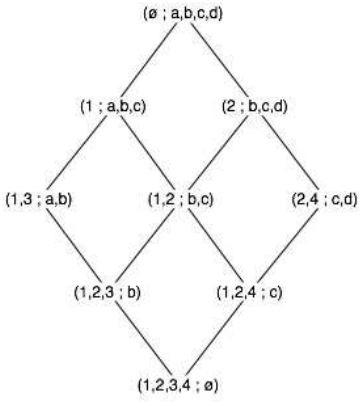

Figure 3. Hass Diagram.

Since a concept lattice can have an exponential size, it is often more interesting to build relevant concepts only. In other words, given an attribute set $X \subseteq A$, the purpose is to compute the concept with the smallest Intent containing X. This work is carried out using two operators, f and g defined as follows:

- $f(X) = \{o \in O \mid a \in X, (o, a) \in R\}$,
- $g(Y) = \{a \in A \mid o \in Y, (o, a) \in R\}$.

Intuitively, given a set of attributes, f returns the set of objects that share these attributes. Similarly, g returns the set of attributes shared by a given set of objects. Taking a set of attributes X, we define the concept associated to X by $(f(X), g(f(X))$. Note that f and g play a symmetric role, consequently the concept induced by a set of objects Y can be defined by $(f(g(Y)), g(Y))$. These composite functions $f(g())$ and $g(f())$ are called the closure operators of the concept lattice. Hence, for a set $X \subseteq A$ (resp. $Y \subseteq O$), the set $g(f(X))$ (resp. $f(g(Y))$) is called the closed set of X (resp. Y). For instance, we can consider the set $\{a\}$ from the previous example. We have $f(\{a\}) = \{1, 3\}$ and $g(\{1, 3\}) = \{a, b\}$, we deduce the concept with the smallest Intent containing $\{a\}$: $(1, 3; a, b)$. As another example, given the set $\{b, d\}$, the associated concept is $(2; b, c, d)$, indeed $f(\{b, d\}) = \{2\}$ and $g(\{2\}) = \{c, b, d\}$.

Algorithm 1 [7] computes the concept with the smallest Intent containing an attribute set X.

Algorithm 1: $Closure_{\mathbb{K}}(X)$: closure algorithm.

Input: A formal context $\mathbb{K} = (O, A, R)$
Input: A set of attributes $X \subseteq A$
Output: The formal concept $(f(X), g(f(X)))$

```
begin
    E = ∅ ;
    I = A ;
    for o ∈ O do
        if X ⊆ g(o) then
            I = I ∩ g(o) ;
            E = E ∪ o
    return (E, I)
end
```

Algorithm 2 computes the interesting concepts only, that is a fragment of the concept lattice. This algorithm, inspired from [19], enables to construct a set S containing successors particularly immediate successors of a given concept (E, I). It is based on the previous closure algorithm called $Closure_{\mathbb{K}}(X)$.

Algorithm 2: Successors algorithm.

Input: A formal concept (E, I)
Output: A set $Succ((E, I)) \subseteq S$

```
begin
    S = ∅ ;
    for a ∈ A \ I do
        S = S ∪ Closure_K(I ∪ a) ;
    return S
end
```

The closure algorithm has a complexity of $\mathcal{O}(m * n)$ where m is the number of objects and n the number of attributes. The successors algorithm costs $\mathcal{O}(n^2 * m)$. Note that Algorithm 2 computes at most n concepts. The two previous algorithms allow to compute relevant concepts without having to build the concept lattice.

FCA is a classification method allowing to organize objects according to their attributes. The Extent of a formal concept (E, I) defines an equivalence class of objects which share the same set of attributes I. The closure algorithm computes this class from a subset of I. Compared to brute-force approaches for selection, we can construct equivalence classes without knowing exactly all the shared attributes. The equivalence class E can be refined using the successors algorithm, which returns a set of immediate successors of (E, I). This set forms a subset of the concept lattice and it can be ordered as a tree. This tree can be used as a decision-making tool for selection.

3.2 Application

The main challenge of our approach is to select at runtime the concrete services, given a design model. The available services are stored in a dynamic runtime model (arrival and departure of services). The idea is to adapt the theoretical framework of FCA to service domain.

To do so, the services registry can be viewed as a formal context, as illustrated by Table 3, where:

- the concrete services are the objects,
- and, the service types, functional and non-functional properties, QoS, reputation are the attributes.

	$t_1, ..., t_i$	$f_1, ..., f_j$	$nf_1, ..., nf_k$
s_1			
...			
s_n			

Table 3. Runtime Model.

In the design model, the user expresses the specifications with two categories of properties (attributes): mandatory and optional. The mandatory attributes are the characteristics required in the specifications: functionality, list of feasible technologies, and some non-functional properties. For example, the selected service Temperature must give temperatures, be implemented with UPnP or DPWS technologies and require an authentication by login/password.

We propose to compute the concept associated to this specification with the Closure Algorithm 1. The set of mandatory characteristics is an input of the algorithm. As result, we obtain a formal concept $(g(X), f(g(X)))$ where:

- $g(X)$ is the set of services that share the mandatory properties,
- and $f(g(X))$ is the set of properties that are shared by the service(s) obtained by $g(X)$. In this set of properties, there is at least the mandatory properties and eventually some other properties.

The set of services $g(X)$ can be viewed as an equivalence class. Each service can be a substitute for all other of this set. In the case of a selected service departure, it is easy to substitute it without examining the runtime model again. As previously explained, the equivalence class can be refined with the successors computation. The set of successors is an extract of the concept lattice which can be used as a decision-making tool. The selection of a concrete service is made according to the user preferences by pruning the branches non-relevant.

4 Experiments and results

We present in this section a use case developed by Thales inc. in the European SODA project. It deals with alarm management in emergency situations. The alarm management system can be viewed as a service composition. The system collects data from sensors such as temperatures. These data are gathered, and then recorded. Finally, according to the value of the gathered data, an action is triggered on the system. This alarm management system is a service composition with four activities, illustrated by Figure 4:

- *Temperature* is in charge of collecting temperatures from available sensors;
- *Analysis* performs a computation on the gathered data;
- *Storage* records the data in a database or XML file;
- *Action* triggers the appropriate action when abnormal data are received.

On this alarm system, we add security properties. Integrity and confidentiality properties are added to *Analysis* and integrity property is required to store the analyzed data.

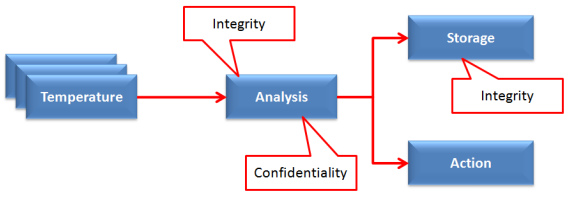

Figure 4. Alarm system.

Table 4 represents the runtime model such as defined in the previous section. Columns are divided into three groups:

- *technologies*: Web Services (WS), UPnP and DPWS;
- *functionalities*: Temperature Sensor (TS), Switch Power (SP), Fan Speed (FS), Analysis (A) and Storage (S);
- *security properties*: Authentication by Username and password (AU), Authentication by X.509 Certificate (AC), Integrity (I) and Confidentiality (C).

	WS	UPnP	DPWS	TS	SP	FS	A	S	AU	AC	I	C
S_1		X		X							X	
S_2		X			X					X	X	X
S_3		X				X				X		
S_4		X		X							X	
S_5		X		X						X	X	X
S_6	X						X	X	X		X	X
S_7	X						X				X	X
S_8	X							X		X	X	
S_9			X	X								
S_{10}			X			X				X		
S_{11}		X			X					X		
S_{12}		X		X						X	X	X
S_{13}			X			X				X	X	

Table 4. Example of runtime model.

In this example, the runtime model does not evolve in order to simplify the example. With this (pre-)defined runtime model , the selection of concrete services consists in running the selection algorithm for each abstract service of the design model. The expected results are presented as a formal concept and its possible successors.

All temperature sensors. The selection consists in the computation of a formal concept, which defines an equivalence class. With the closure algorithm, we obtain the formal concept $(S_1, S_4, S_5, S_9, S_{12}; TS)$ containing the set of services that have the temperature functionality.

Analysis with integrity and confidentiality. The closure algorithm takes as input the set of attributes: Analysis, Integrity and Confidentiality. The formal concept is $(S_6, S_7; A, I, C, WS)$. Note that all services (S_6 and S_7) are Web Services. The difference between the services S_6 and S_7 is that S_6 has an additional functionality: *Storage*. This additional functionality is deduced from the results of the successors algorithm.

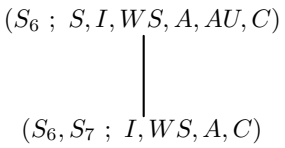

Figure 5. "Analysis" partial lattice.

Storage with integrity. As previously, the formal concept is computed: $(S_6, S_8; S, I, WS)$. Two Web Services provide the storage functionnality with integrity. The successors of this concept are $(S_6; S, I, WS, A, AU, C)$ and $(S_8; S, I, WS, AC)$. This two services provide additional functionalities and security properties. The security properties are disjoint: the authentication mechanism is by username for S_6 while by X.509 Certificate for S_8. This difference aids into the selection of an appropriate service because the user has not all the security information.

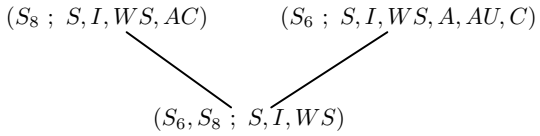

Figure 6. "Storage" partial lattice.

Action. The action requires a Fan Speed service. The formal concept is $(S_3, S_{10}, S_{13}; FS, AC)$ and its successors are $(S_3; FS, AC, UPnP)$ and $(S_{10}, S_{13}; FS, AC, DPWS)$. The selected service necessarily require an authentication by certificate; the user must have a valid certificate to act with the fan. The selected service is an UPnP or a DPWS service. The successor of $(S_{10}, S_{13}; FS, AC, DPWS)$ is $(S_{13}; FS, AC, DPWS, I)$. The service S_{13} ensures the integrity data (received and emitted).

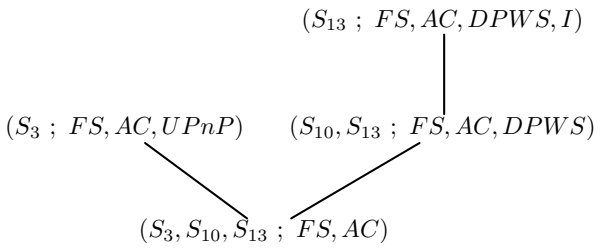

Figure 7. "Action" partial lattice.

We believe that benefits of our approach are:

- *Avoid negative answers of the selection.* If we search, in the context Table 4, an Analysis Web Service with no security properties, it does not exist. The formal concept gives a precision. The formal concept is $(S_6, S_7; WS, A, I, C)$. There are two services with the functionality analysis implemented by Web Service and these services require/provide security properties (integrity and confidentiality).
- *Equivalence classes.* The formal concept contains in the left part an equivalence class of services, *i.e.* all

services have the same characteristics. If one service is required for the application, when a service departure occurs, the others service of the class can also be used. Consequently, reaction time is reduced: the runtime model is queried juste one time per activity specification. This search is done in the size of the runtime model ($\mathcal{O}(n * m)$).

- *Optimization of the search tree building.* The search tree is not built in depth. The nodes are computed dynamically during the search. The aim of selection is to find one or more services meeting given characteristics. In the case of many appropriate services (*i.e.* they are in the same equivalence class), the successors are built to remove selected services offering additional features which can be prohibitive, *e.g.* in our example the formal concept is (S_3, S_{10}, S_{13};FS, AC) and its successors are (S_3;$FS, AC, UPnP$) and (S_{10}, S_{13};$FS, AC, DPWS$). One of these two branches can be pruned according to the options specified by the user and thus limit the choice of selected services.

- *Minimize the security constraints.* The formal concept contains the set of services which have the same characteristics. But these services can have more characteristics. In the case of security, it is interesting to be only limited to security specifications. The computation of successors allows defining the difference between services. For example, the computation of the successors of the formal concept for the activity Action of the alarm system shows that the service S_{13} requires/provides more security concepts (integrity).

5 Related work

There are many approaches to select services based on criteria and constraints of QoS. Conventional approaches propose to use brute-force-like algorithms to obtain the best service composition (according to QoS utility functions). Channa *et al.* [3] propose to solve the selection problem by using an approach based on constraint satisfaction. They present a discovery engine associated to a constraint optimizer which selects Web Services that are optimal and satisfy client's constraints. Another approach [10] uses genetic algorithms. These algorithms define some QoS utility functions and try to maximize them. One disadvantage of this kind of algorithm is that it can run indefinitely. To avoid this, some constraints must be predefined. Therefore these algorithms don't take into account some constraints inherent in the context of dynamic service environments. A selected composition can be unusable at runtime (service unavailable, change in QoS...). Other approaches propose to consider this dynamic aspect by not selecting a configuration but a description from which the best composition is selected at runtime. Estublier *et al.* [6] propose to extend the concept of composite (configuration) to represent the application along the different life-cycle phases (from design to execution). Mabrouk *et al.* [13] present an algorithm taking account into the concept of dynamic binding allowing composition with on-the fly services.

Azmeh *et al.* propose a tool named WSPAB [1] that aims to define a complete solution for facilitating the task of finding the most pertinent Web Service. The WSPAB tool select automatically services by filtering Web Services according to certain aspects of QoS and certain user requirements. It classifies the filtered services using the FCA approach, enabling users to easily select their needed service. It is possible with their approach to also identify potential service substitutes and to keep trace of them for future use. The principle of this method is very similar to our approach, but it is only applied to Web Services. However, the filtration criteria are hardly applicable to domains where the selection is based on the maximum number of service operations for each Web Service. Web Services signatures are extracted from WSDL and sorted according to the number of input parameters. This gives groups of operation signatures to construct the concept lattice and the equivalent concepts. So, the similarity factor between the operations is the number of input parameters. It is not always relevant to use this criterion because the functionality of the service is lost while it is a necessary element to use for selecting the appropriate service.

6 Conclusion

In pervasive environments, the selection of the right services to achieve desired functions, expressed in abstract terms, is a major issue. Solving it requires to select the most appropriate service given a set of available services but also to do it in time. Indeed, in many pervasive applications, time is an important parameter.

In previous works, we observed that, in many industrial use cases, brute force like algorithms for services selection are not effective. They are too costly and not adapted to situations where constraints may be released. In this paper, we propose to structure the services registry with an approach based on Formal Concept Analysis. Our purpose is to speed up the selection process and to improve decision making through the building of a concept lattice. The complexity of such computation is in the order of brute force algorithms. But, it can be reused to perform more complex searches, where constraints are changed. In this situation, the equivalence classes and successors avoid reiterate each time the selection algorithm which significantly improves performance at runtime.

In the current version, the concept lattice is re-built on every service departure or arrival. We are currently working on an improved version where the lattice is built only on relevant changes in the environment, that is services implied in current applications.

References

[1] Z. Azmeh, M. Huchard, C. Tibermacine, C. Urtado, and S. Vauttier. WSPAB: A Tool for Automatic Classification & Selection of Web Services Using Formal Concept Analysis. In *ECOWS '08: Proceedings of the 2008 Sixth European Conference on Web Services*, pages 31–40, Washington, DC, USA, 2008. IEEE Computer Society.

[2] P. Cellier. Formal concept analysis applied to fault localization. In Robby, editor, *International Conference on Software Engineering (ICSE 2008) Companion*, pages 991–994. ACM, 2008.

[3] N. Channa, S. Li, A. W. Shaikh, and X. Fu. Constraint satisfaction in dynamic web service composition. *Database and Expert Systems Applications, International Workshop on*, 0:658–664, 2005.

[4] S. Chollet and P. Lalanda. An Extensible Abstract Service Orchestration Framework. In *ICWS '09: Proceedings of the 2009 IEEE International Conference on Web Services*, pages 831–838, Washington, DC, USA, 2009. IEEE Computer Society.

[5] C. Escoffier, R. S. Hall, and P. Lalanda. iPOJO: an Extensible Service-Oriented Component Framework. *IEEE International Conference on Services Computing (SCC 2007)*, pages 474–481, July 2007.

[6] J. Estublier, I. Dieng, E. Simon, and G. E. Vega Baez. Flexible composites and automatic component selection for service-based applications. In *Proceedings of the 4th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE)*, Milan, Italy, may 2009. INSTICC Press.

[7] B. Ganter and K. Reuter. Finding all closed sets: A general approach. *Order*, 8(3):283–290, 1991.

[8] B. Ganter and R. Wille. *Formal Concept Analysis - Mathematical Foundations*. Springer, 1999.

[9] IBM. Web Services Atomic Transaction (WS-AtomicTransaction), November 2004. `http://download.boulder.ibm.com/ibmdl/pub/software/dw/library/ws-atomictransaction200411.pdf`.

[10] M. C. Jaeger and G. Mhl. Qos-based selection of services: The implementation of a genetic algorithm. In *In KiVS 2007 Workshop: Service-Oriented Architectures und ServiceOriented Computing (SOA/SOC*, pages 359–370, 2007.

[11] F. Jammes, A. Mensch, and H. Smit. Service-Oriented Device Communications Using the Devices Profile for Web Services. In *MPAC'05: Proceedings of the 3rd international workshop on Middleware for pervasive and ad-hoc computing*, pages 1–8, New York, NY, USA, 2005. ACM.

[12] P. Lalanda and C. Marin. A Domain-Configurable Development Environment for Service-Oriented Applications. *IEEE Software*, 24(6):31–38, 2007.

[13] N. B. Mabrouk, S. Beauche, E. Kuznetsova, N. Georgantas, and V. Issarny. Qos-aware service composition in dynamic service oriented environments. In *Middleware '09: Proceedings of the 10th ACM/IFIP/USENIX International Conference on Middleware*, pages 1–20, New York, NY, USA, 2009. Springer-Verlag New York, Inc.

[14] OASIS. Web Services Security: SOAP Message Security 1.0, March 2004. `http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf`.

[15] OASIS. Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0, March 2005. `http://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf`.

[16] OASIS. eXtensible Access Control Markup Language (XACML) Version 2.0, February 2005. `http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-core-spec-os.pdf`.

[17] OSGi Alliance. OSGi Service Platform Core Specification, Release 4, Version 4.1, April 2007. `http://www.osgi.org/download/r4v41/r4.core.pdf`.

[18] M. P. Papazoglou. Service-Oriented Computing: Concepts, Characteristics and Directions. In *WISE'03: Proceedings of the Fourth International Conference on Web Information Systems Engineering*, pages 3–12, Washington, DC, USA, 2003. IEEE Computer Society.

[19] N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal. Discovering frequent closed itemsets for association rules. In *ICDT '99: Proceedings of the 7th International Conference on Database Theory*, pages 398–416, London, UK, 1999. Springer-Verlag.

[20] D. Poshyvanyk and A. Marcus. Combining formal concept analysis with information retrieval for concept location in source code. In *Proceedings of the 15th IEEE Int'l Conf. on Program Comprehension*, pages 37–48. IEEE CS, June 2007.

[21] V. Snasel, Z. Horak, J. Kocibova, and A. Abraham. Analyzing social networks using fca: Complexity aspects. In *WI-IAT '09: Proceedings of the 2009 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology*, pages 38–41, Washington, DC, USA, 2009. IEEE Computer Society.

[22] W3C. XML Encryption Syntax and Processing, December 2002. `http://www.w3.org/TR/xmlenc-core/`.

[23] W3C. XML Signature Syntax and Processing (Second Edition), June 2008. `http://www.w3.org/TR/xmldsig-core/`.

[24] J. Yu, P. Lalanda, and S. Chollet. Development Tool for Service-Oriented Applications in Smart Homes. In *SCC '08: Proceedings of the 2008 IEEE International Conference on Services Computing*, pages 239–246, Washington, DC, USA, 2008. IEEE Computer Society.

[25] E. Zeeb, A. Bobek, H. Bohn, and F. Golatowski. Service-Oriented Architectures for Embedded Systems Using Devices Profile for Web Services. In *AINAW '07: Proceedings of the 21st International Conference on Advanced Information Networking and Applications Workshops*, pages 956–963, Washington, DC, USA, 2007. IEEE Computer Society.